Exercise 1

Juuso Korhonen - 652377
ELEC-E8125 - Reinforcement Learning

September 11, 2022

1 Task 1

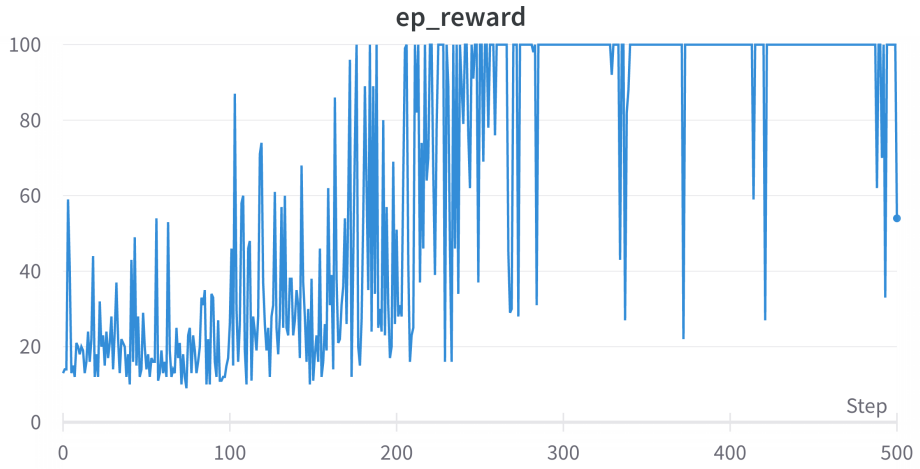

Figure 1: Train episode reward.

Average test reward: 927.8 episode length: 927.8.

1.1 Question 1.1

As we can see from table 1, none of them learn to always balance the pole for 1000 timesteps. This is likely due to 100 timesteps not being enough to test if the system has learned to actually stabilize the pole. Our results seem to imply that.

2 Task 2

2.1 Question 2.1

As we can see from table 2, the performance is wildly different with different seeds. Some seeds like 2 and 1000 manage to get perfect average test reward, where as rest of the seeds

Seed	Average test reward
10	893.7
100	853.5
1000	828.0
10,000	990.4
100,000	687.3

Table 1: Average testing rewards with 5 different seeds used in testing.

Seed	Average test reward (after training with same seed)
10	373.1
2	1000.0
100	433.6
1000	1000.0
10,000	637.4

Table 2: Average testing rewards with 5 different seeds used for training/testing cycle.

result in poor test performance. This might be because e.g. if the seed is used to initialize the weights of the model, the model might get stuck in local optimum.

2.2 Question 2.2

The implications of this stochasticity of the test performance depending on the set seed are that when comparing different models, one should take this into account e.g. by doing the training and testing with different seeds to get accurate estimation of the results other users might get.

3 Task 3

Reward function for behavior (1):

```
def get_reward(self, prev_state, action, next_state):
        # TODO: Task 3: Implement and test two reward functions
        ########## Your code starts here ##########
        # A dummy reward, replace it with yours.
        prev_state_theta0 = prev_state[0]
        next_state_theta0 = next_state[0]

        diff = next_state_theta0 - prev_state_theta0
        return -(0.2-diff)**2

        ########## Your codes end here ##########
```

Reward function for behavior (2):

```
def get_reward(self, prev_state, action, next_state):
        # TODO: Task 3: Implement and test two reward functions
        ########## Your code starts here ##########
        # A dummy reward, replace it with yours.
        #prev_state_cart = self.cartesian_pos(prev_state)
        next_state_cart = self.get_cartesian_pos(next_state)

        diff = np.linalg.norm(self.goal-next_state_cart)
        return -diff

        ########## Your codes end here ##########
```

4 Task 4

4.1 Question 4.1

Highest reward is achieved at point: (1,1) (cartesian). Lowest reward is achieved when the euclidean is the highest from point (1,1).

4.2 Question 4.2

Inspecting both the state reward plot 2 and policy plot 3, we can see that optimal way to reach goal state is not always achieved (in number of steps). For example from state (J1: -3.14, J2: 0) it would be obviously much more efficient to go into -J2 direction, but policy plot indicates that the optimal action would be to +J1.

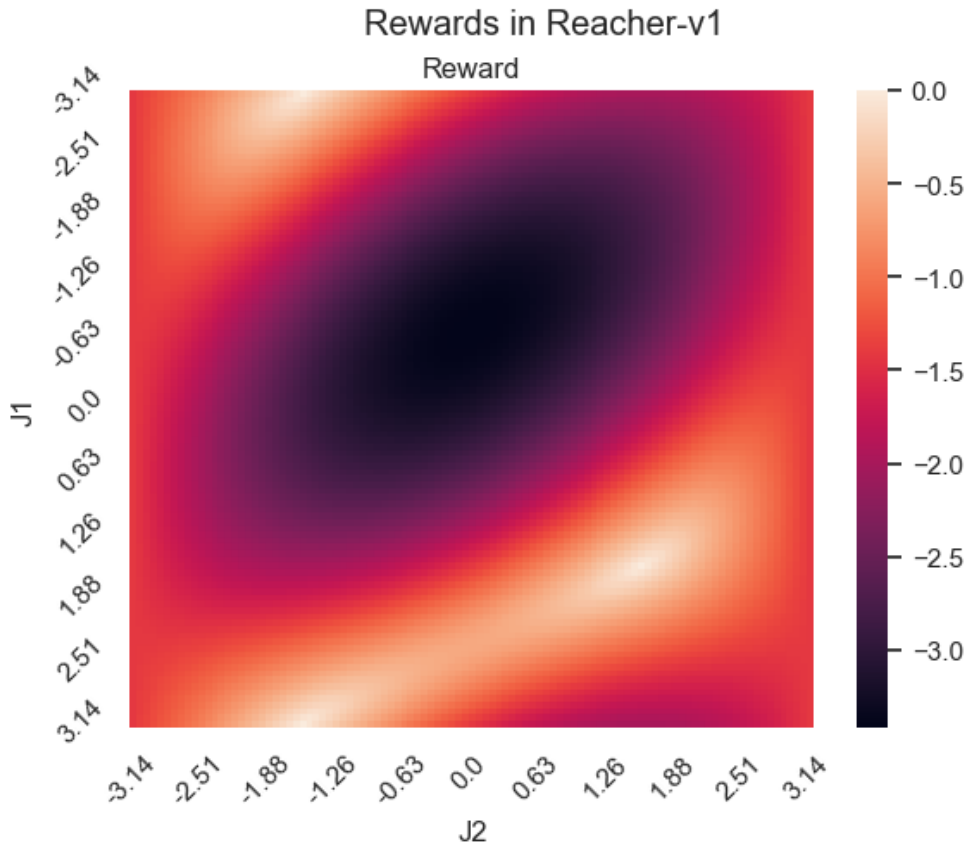

Figure 2: State reward function plot.

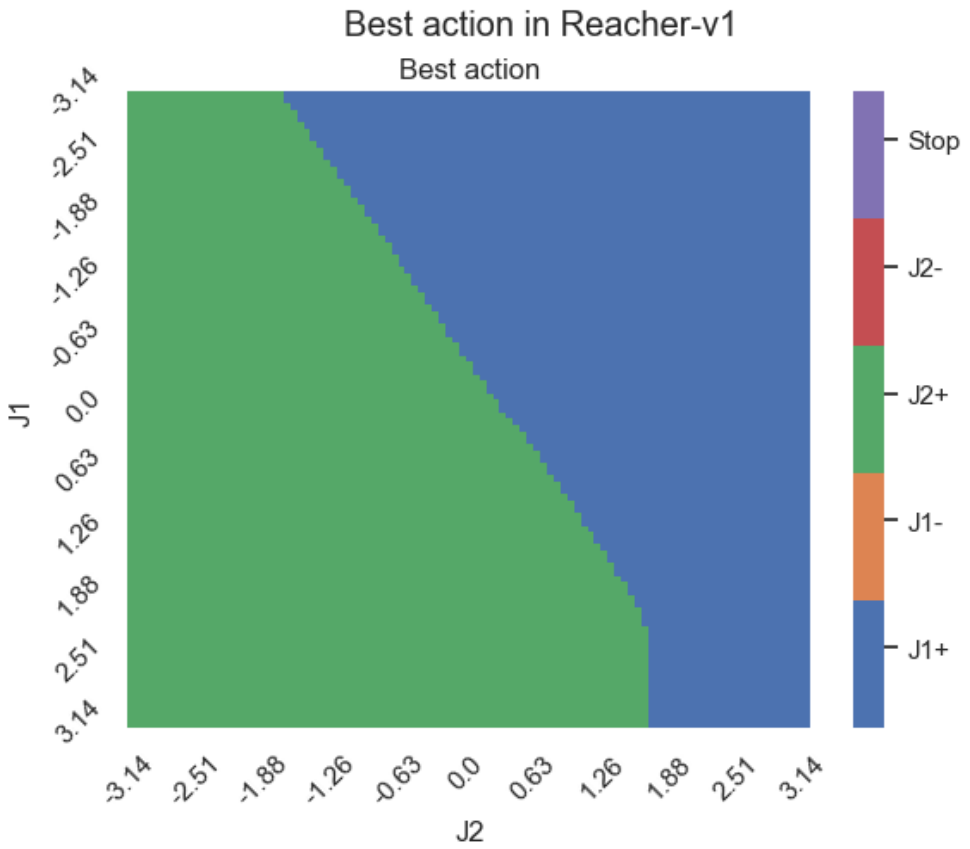

Figure 3: Policy plot.